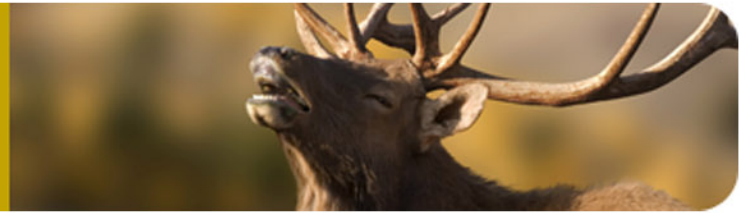




J-Fall

2 november 2011 - Hart van Holland



## Cassandra in production at eBuddy

Joost van de Wijgerd



## About this talk

- Intro
- Cassandra overview
- Use cases at eBuddy
  - Data model
  - Coding samples
- Production Environment
- Next Steps for Cassandra at eBuddy
- Questions



## About me

- 35 years old
- Living in Amsteram with my wife and two kids
- Co-Founder of JTeam (now Dutchworks)
- Co-Founder of SpringSource
- Independent Consultant
- Angel Investor
- Since 2009 Backend Architect at eBuddy



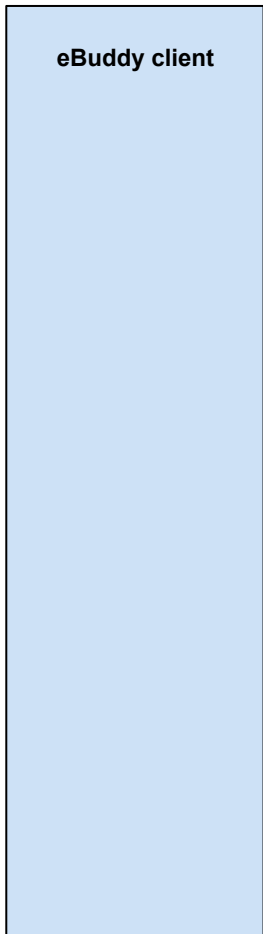
## About eBuddy

- IM network aggregator on Web and Mobile
- More than 250 Million registered users
- Over 30 Million monthly active users
- 16 Billion messages handled per month
- New Mobile Messaging product: **XMS**
- All important backend services are Java



eBuddy Client component

eBuddy public server-side services



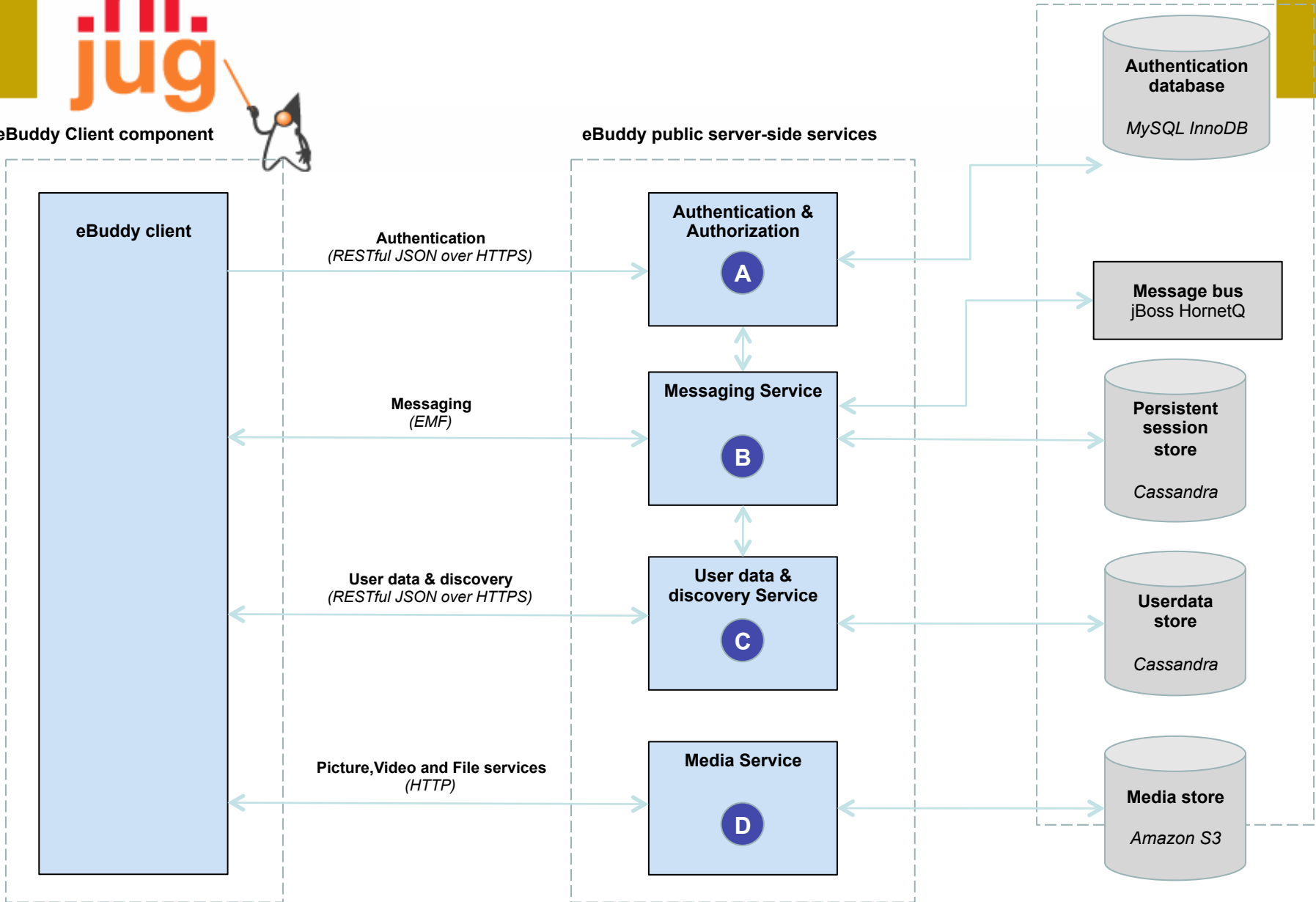
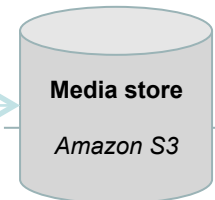
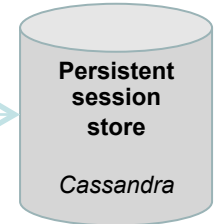
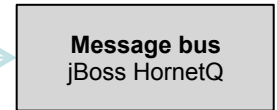
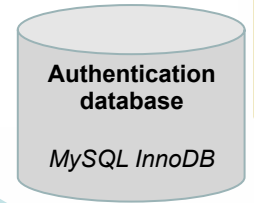
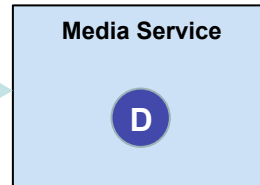
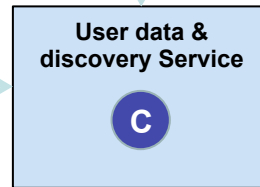
eBuddy client

Authentication  
(RESTful JSON over HTTPS)

Messaging  
(EMF)

User data & discovery  
(RESTful JSON over HTTPS)

Picture, Video and File services  
(HTTP)





## What is Cassandra?

- Data Model
  - Column Oriented
  - No predefined schema
  - Denormalized
- Deployment Model
  - Multiple row replicas, No master
- Data Consistency
  - Eventually Consistent, CAP Theorem



## Cassandra Data Model

- Key-Value Store with Maps as Values
  - Keyspace with (Super)ColumnFamilies
    - Analogous to database with tables
- Multi level Map
  - `Map<byte[],OrderedMap<byte[],byte[]>>`
    - a.k.a ColumnFamily
  - `Map<byte[],OrderedMap<byte[],OrderedMap<byte[],byte[]>>>`
    - a.k.a SuperColumnFamily
- Give meaning to `byte[]` in code and config

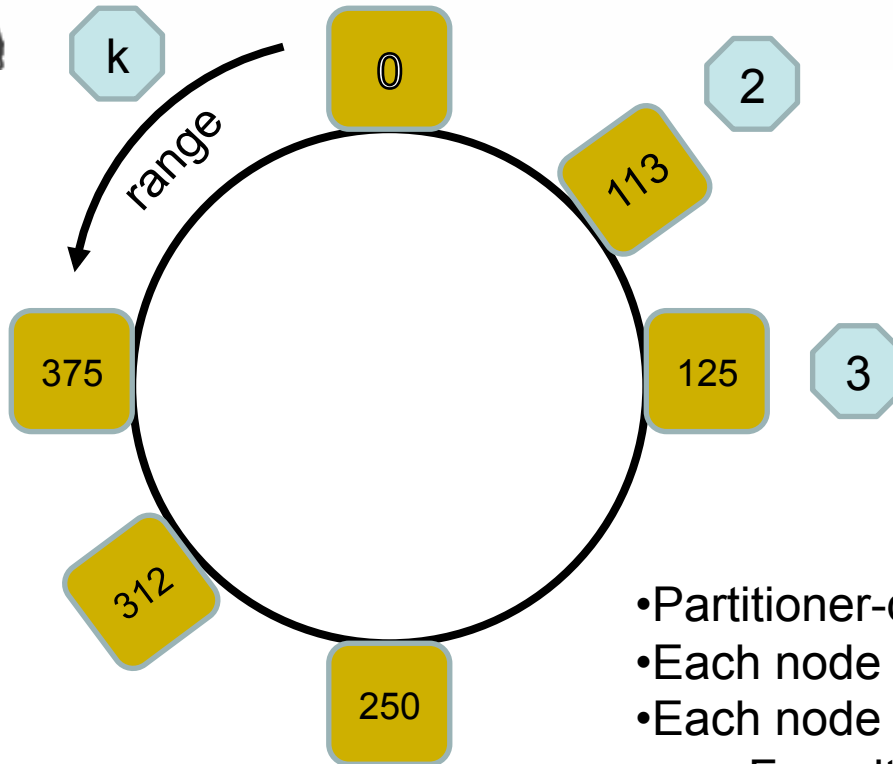


## Cassandra Deployment Model

- Cassandra Nodes form a Ring
- Each key is mapped to a position on the Ring
- Each key can (should) be replicated
  - Determined by replication factor
- Keys are mapped to Nodes:
  - Based on Partitioner implementation
  - Based on Replication Strategy
- Replication factor of 3 and 4 nodes are recommended minimum for production use



## The Ring



### Tokens:

- Partitioner-dependent
- Each node has unique token
- Each node claims a range
  - From its token to token of previous node on the ring



## (Eventually) Consistent

- Brewer's CAP Theorem:
  - Consistency
  - Availability
  - Partition tolerance
- Cassandra values Availability and Partition tolerance
- Consistency is tunable at various levels
  - ONE, QUORUM and ALL
- No Row Locking



## Write Path

- Client writes to one Coordinator Node in the cluster
- Coordinator Node sends write to all Replicas
- Each Replica first writes to disk commit log
- (Over-)writes data in memory (Memtable)
- Memtable flushed to disk (SSTable) at regular intervals
  - Time, Throughput or Amount of Columns



## Read Path

- Client request read on Coordinator Node
- Coordinator Node determines replicas and requests data
  - One node sends actual data
  - Other node(s) send digest
- Each replica first checks Memtable then SSTable(s)
  - Uses bloom filter to determine if SSTable has row fragment



## Why use Cassandra at eBuddy?

- Flexible datastore for (mobile) clients and backend services
- We need to store more and more data
- Leverage existing (Java) knowledge
- Easy to monitor and maintain for Operations
- Fault tolerant and no SPOF
- MySQL doesn't scale very well!



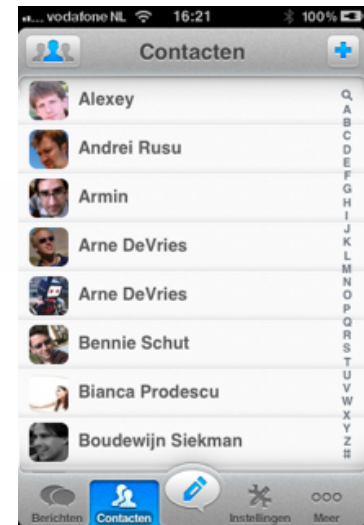
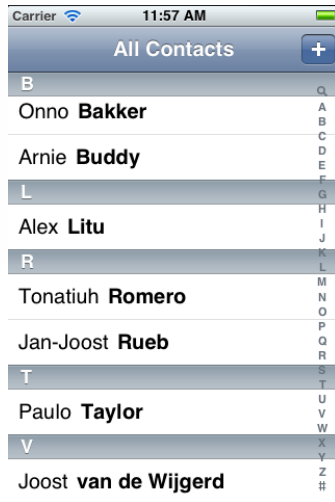
## What do we use Cassandra for?

- User Discovery Service
  - Populate contact lists based on source *Social Graphs* (e.g. Facebook friends)
- User Data Service
  - User Demographics and Preferences
  - Contact Lists
  - User specific Service Settings
- Persistent Session Store
  - Serialized sessions for failover and scale out



## User Discovery

- What it does
  - Discover contacts by processing social graphs such as address books and facebook friend lists
  - Works on any kind of "social" network graphs
- Algorithm
  - Process uploaded social graphs
  - Discover new connections
  - Inform users about new connections



Discovery







## Data Model: ServiceSettings

- SuperColumnFamily
  - Key → unique User Id (UUID)
  - SuperColumnName → Service Name
  - Columns → SocialGraph keys for this user
- User Discovery Service uses it as an Index into SocialGraph



## Data Model: ServiceSettings

```
{ "2c8e1940-352f-11e0-b686-0030488df8a2": {  
  "uds" : {  
    "xms_2c8e1940-352f-11e0-b686-0030488df8a2@xms.ebuddy.com": "",  
    "fb_805048253": "",  
    "tel_K2lvAUMIV40JRrENMYniTg": "",  
    "xms_2c8e1940-352f-11e0-b686-0030488df8a2@xms-beta.ebuddy.com": "",  
    "pin_EBDDFF4C": ""  
  }  
}}
```



## Data Model: SocialGraph

- SuperColumnFamily
  - Key → *Network Identity* e.g. fb\_128367622
  - SuperColumnName → Network Id of contact
  - Columns → Contact details
- Special SuperColumn *UUUID* that contains the user id as a column name
- List of contacts for a social graph of an eBuddy user and associated properties



## Data Model: SocialGraph

```
{ "xms_2c8e1940-352f-11e0-b686-0030488df8a2@xms.ebuddy.com": {  
  "UUUID": {  
    "2c8e1940-352f-11e0-b686-0030488df8a2": ""  
  },  
  "xms_0e0ac8f0-35b7-11e0-b686-0030488df8a2@xms.ebuddy.com": {  
    "facebook_id": "768365191",  
    "name": "Paulo Taylor",  
    "registered_name": "Paulo Taylor",  
    "sources": [ "tel_PWimuWOAEwcamGptvkvgOw", "fb_768365191" ],  
    "status": "AUTO_ADDED",  
    "type": "USER"  
  },  
  ...  
}
```



## Data Model: SocialGraphInverse

- SuperColumnFamily
  - Key → *Network Identity* e.g. fb\_128367622
  - SuperColumnName → UUID of eBuddy user
  - Columns
    - Names: Network Identity of eBuddy user
    - Values: JSON objects with properties
- List of eBuddy users that are linked to this *Network Identity*



## Data Model: SocialGraphInverse

```
{ "tel_GQxpqUKaxYWrYfNORCASHw": {  
  "61973fc0-35e5-11e0-baab-003048d3746c": {  
    "tel_-Me9JbiEXxNcq-z_Sjgq3w": {  
      "name": "Onno Bakker",  
      "facebook_id": "100003929116324",  
      "type": "USER"  
    }  
  },  
  "0e0ac8f0-35b7-11e0-b686-0030488df8a2": {  
    "tel_b1EzXzi0IF-7J0QuQBtwcw": {  
      "name": "Paulo Taylor",  
      "type": "USER"  
    }  
  }  
}
```



## Working with Cassandra from Java: Introducing Hector

- Abstraction of underlying Thrift layer
- Load balancing and failover between Cassandra nodes
- Mutator object for single or batch updates
- Serializers for Row Keys and (Super)Column names and values



## Hector API: Spring Beans

```
<bean id="roundRobinBalancingPolicyBugFix"  
  class="com.ebuddy.cassandra.hectorbugfix.RoundRobinBalancingPolicyBugFix"/>
```

```
<bean id="cassandraHostConfigurator" class="me.prettyprint.cassandra.service.CassandraHostConfigurator">  
  <constructor-arg index="0" value="10.31.0.14:9160,10.31.0.15:9160,10.31.0.16:9160,10.31.0.17:9160"/>  
  <property name="maxActive" value="128"/>  
  <property name="maxIdle" value="1"/>  
  <property name="exhaustedPolicy" value="WHEN_EXHAUSTED_FAIL"/>  
  <property name="maxWaitTimeWhenExhausted" value="1000"/>  
  <property name="loadBalancingPolicy" ref="roundRobinBalancingPolicyBugFix"/>  
</bean>
```

```
<bean id="cluster" class="me.prettyprint.cassandra.service.ThriftCluster">  
  <constructor-arg value="DEV Cluster"/>  
  <constructor-arg ref="cassandraHostConfigurator"/>  
</bean>
```

```
<bean id="userDataKeyspace" class="me.prettyprint.hector.api.factory.HFactory"  
  factory-method="createKeyspace">  
  <constructor-arg value="UserData"/>  
  <constructor-arg ref="cluster"/>  
</bean>
```



# Hector API: writing to a SuperColumnFamily

```
addToServiceSettings("0e4e40d6-cd7c-4fe2-80ec-bb9bc2ee1015",  
                    "fb_2726388229","xms_0e4e40d6-cd7c-4fe2-80ec-bb9bc2ee1015@xms.ebuddy.com");
```

```
List<HColumn<String,String>> columns = Arrays.asList(  
    HFactory.createColumn(sourceNetworkId,  
        null,  
        StringSerializer.get(),  
        StringSerializer.get()),  
    HFactory.createColumn(targetNetworkId,  
        null,  
        StringSerializer.get(),  
        StringSerializer.get()));
```

```
HSuperColumn<String,String,String> superColumn = HFactory.createSuperColumn("uds",  
    columns,  
    StringSerializer.get(),  
    StringSerializer.get(),  
    StringSerializer.get());
```

```
Mutator<String> mutator = HFactory.createMutator(userDataKeyspace, StringSerializer.get());
```

```
mutator.insert(userId,"ServiceSettings",superColumn);
```



## Hector API: reading SocialGraphInverse

```
public void processSocialGraphInverse(String networkId) {

    SuperSliceQuery<String,UUID,String,Map<String,Object>> query =
        HFactory.createSuperSliceQuery(userDataKeyspace,
                                        StringSerializer.get(),
                                        UUIDSerializer.get(),
                                        StringSerializer.get(),
                                        JacksonSerializer.get());

    query.setKey(networkId).
        setColumnFamily("SocialGraphInverse").
        setRange(null, null, false, Integer.MAX_VALUE);

    QueryResult<SuperSlice<UUID,String,Map<String,Object>>> queryResult = query.execute();

    for (HSuperColumn<UUID, String, Map<String, Object>> superColumn :
        queryResult.get().getSuperColumns()) {
        crossReferenceContact(superColumn);
    }
}
```



# Hector API: reading all user ids from SocialGraph using multiget

```
public List<String> readAllUUIDS(List<String> keys) {
    MultigetSubSliceQuery<String,String,String,String> query =
        HFactory.createMultigetSubSliceQuery(userDataKeyspace,
            StringSerializer.get(),
            StringSerializer.get(),
            StringSerializer.get(),
            StringSerializer.get());

    query.setKeys(keys).
        setColumnFamily("SocialGraph").
        setSuperColumn("UUUID").
        setRange(null, null, false, 1);

    QueryResult<Rows<String,String,String>> result = query.execute();
    List<String> uuids = new ArrayList<String>(result.get().getCount());

    for (Row<String,String,String> row : result.get()) {
        String key = row.getKey();
        HColumn<String,String> uuidColumn = row.getColumnSlice().getColumns().get(0);
        uuids.add(uuidColumn.getName());
    }
}
```

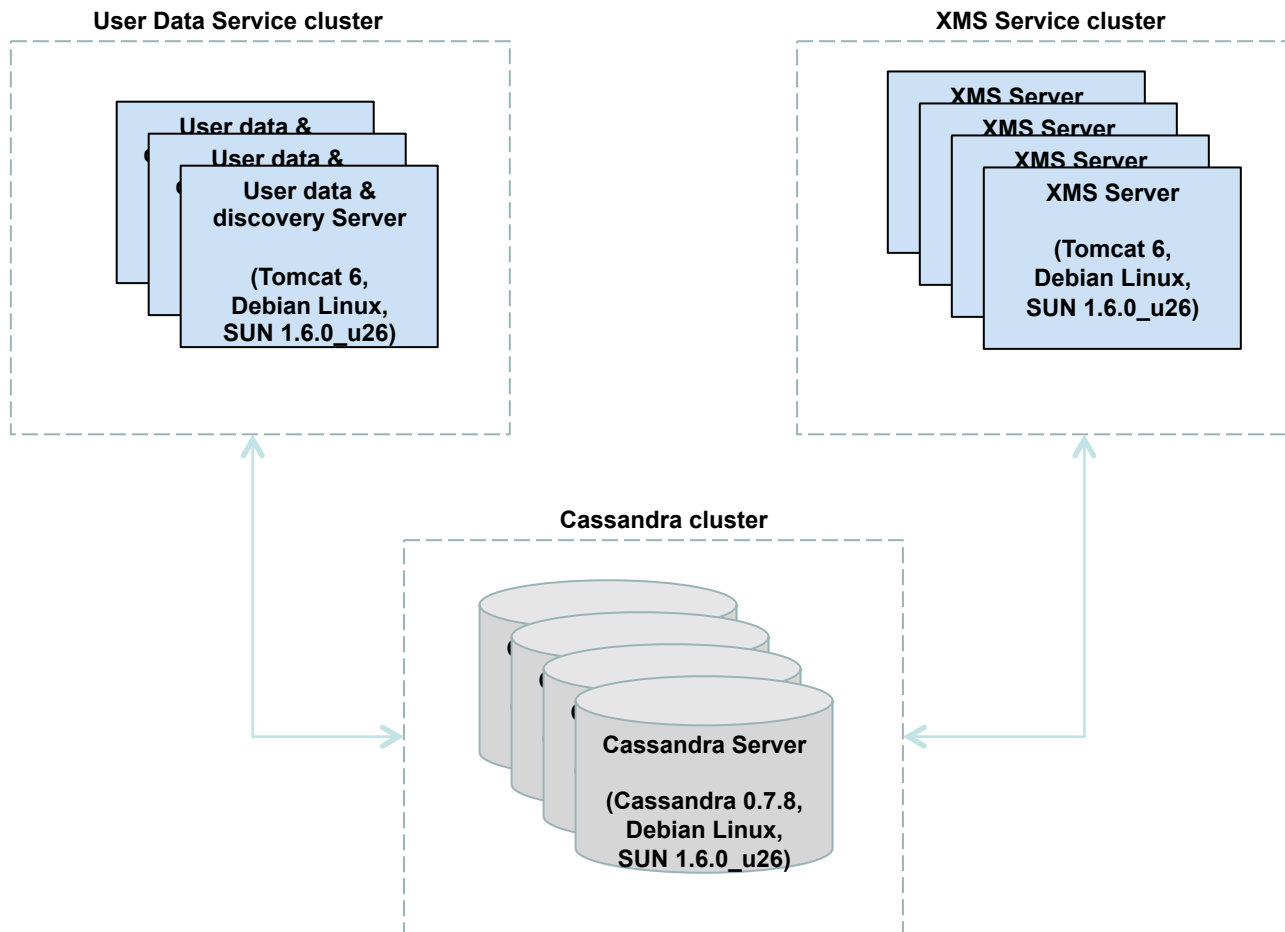


## User Discovery: Algorithm

- Input
  - Uploaded Address Book or Facebook Friends
- Steps
  - AddUserToUserData
  - FindInitialContacts
  - ProcessSourceSocialGraph
- Output
  - Contact List updates



# Production Environment





## Hardware (x4)

- 48GB
- 200 MB Commitlog volume – RAID1
- 2TB Data volume – RAID10
- 16 processor cores
- Sun JVM 1.6.0\_u26
- 16GB Heap
  - Large pages enabled
- Disabled Linux I/O scheduling
  - Huge performance boost for SSTable reads



## Configuration

- Cassandra 0.7.8
- Replication factor = 3
- Key Cache Size 1 Million per CF
- Row Cache disabled
- Memtable max age 1 day
- QUORUM read and write consistency



## Maintenance

- One backup per day per machine
  - Copy to NFS mount, might become unfeasible with more data
- Major Compaction once a day per machine
  - Scheduled to run at different time slots
- Anti-Entropy repair once a week per machine
  - Needed to keep data consistent within the cluster
  - Running on different days of the week

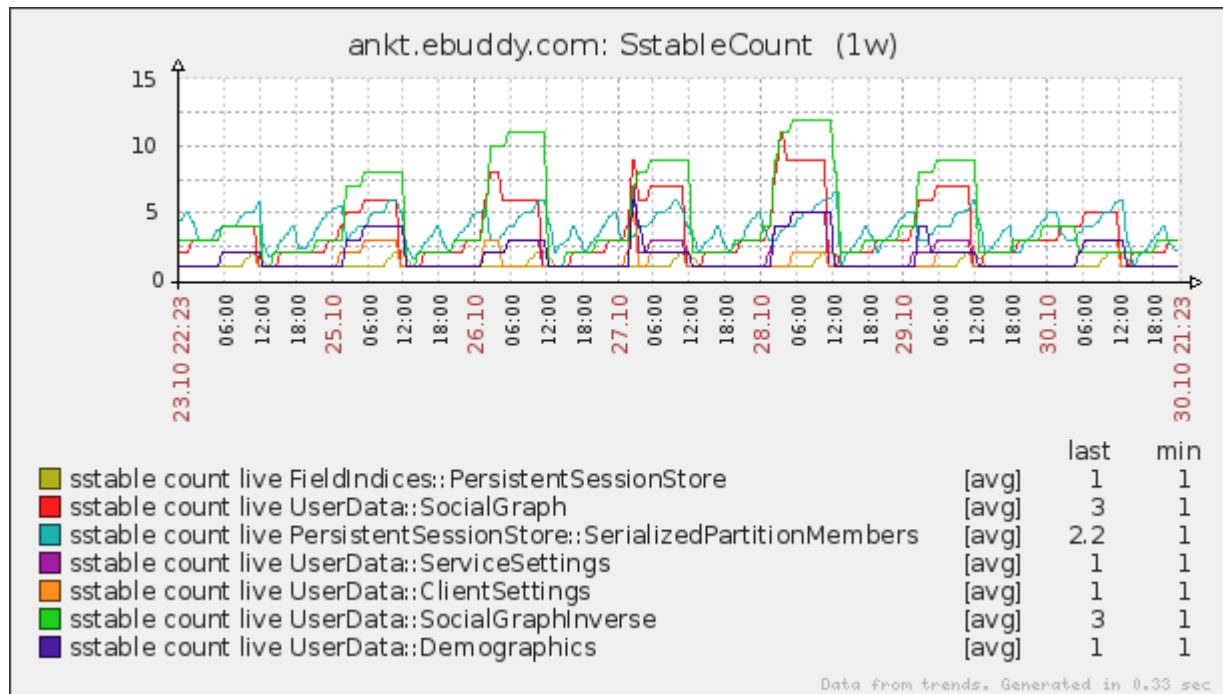


## Some statistics

- Current size of data ~390GB
- ~130 GB actual data
- ~6.4 Million keys in SocialGraph
- ~368 Million keys in SocialGraphInverse

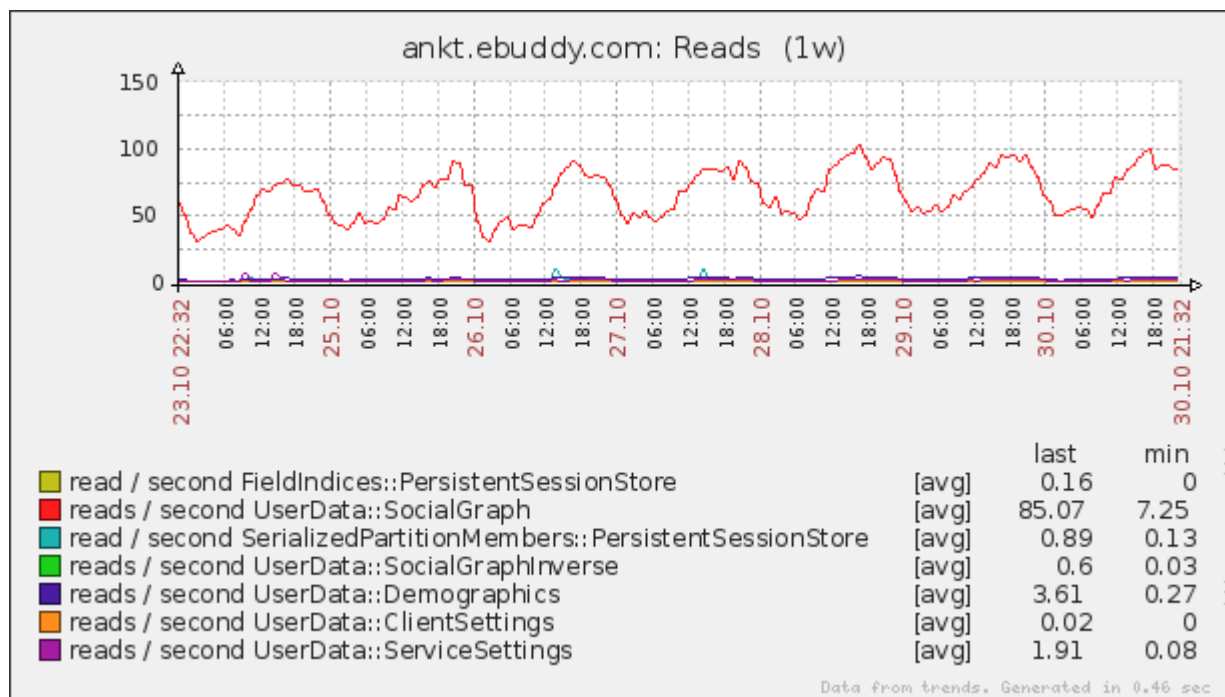


# SSTable Count



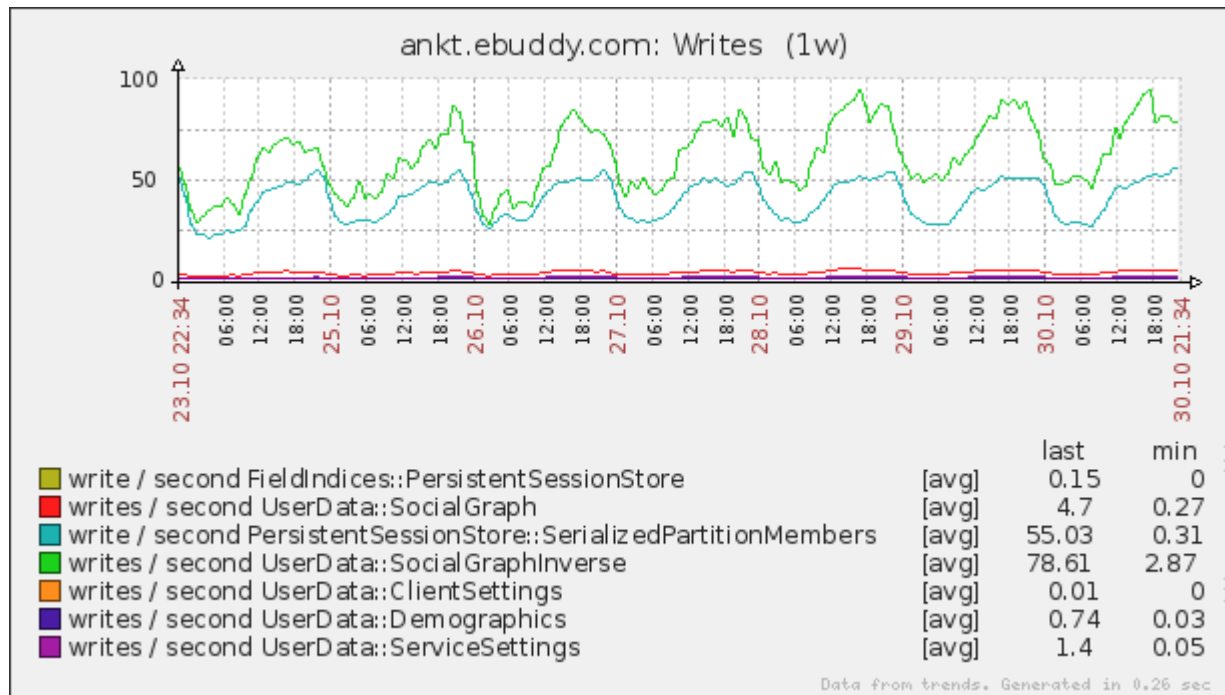


# Reads per second



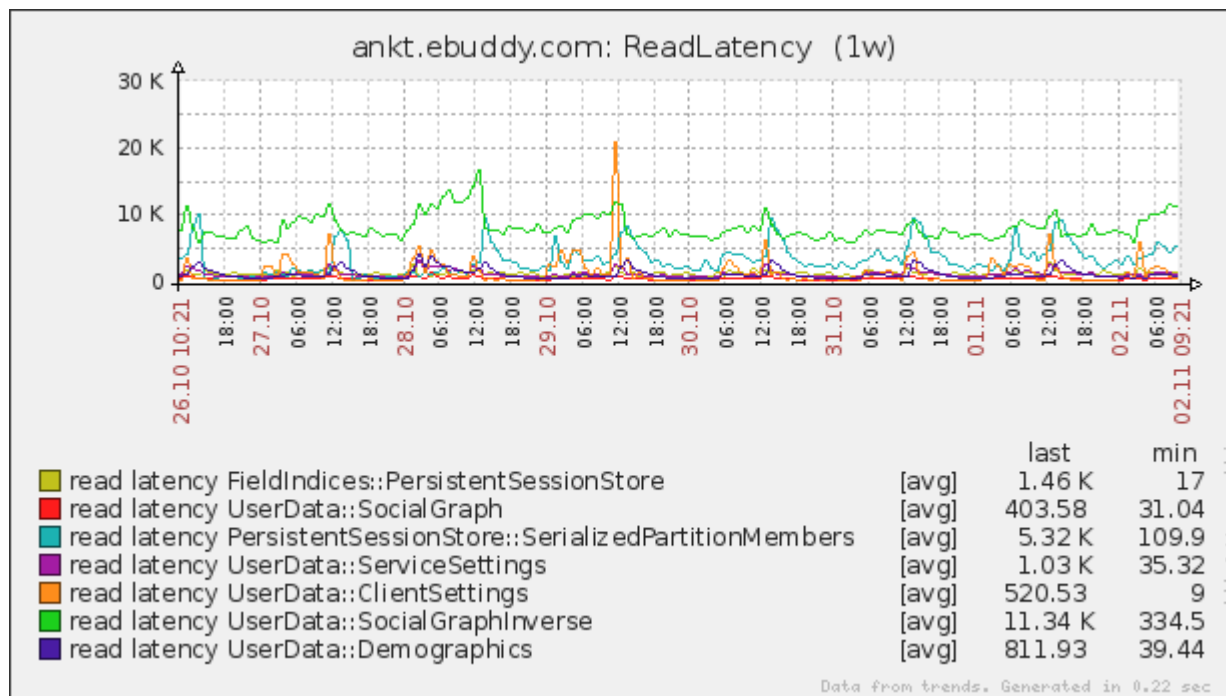


# Writes per second



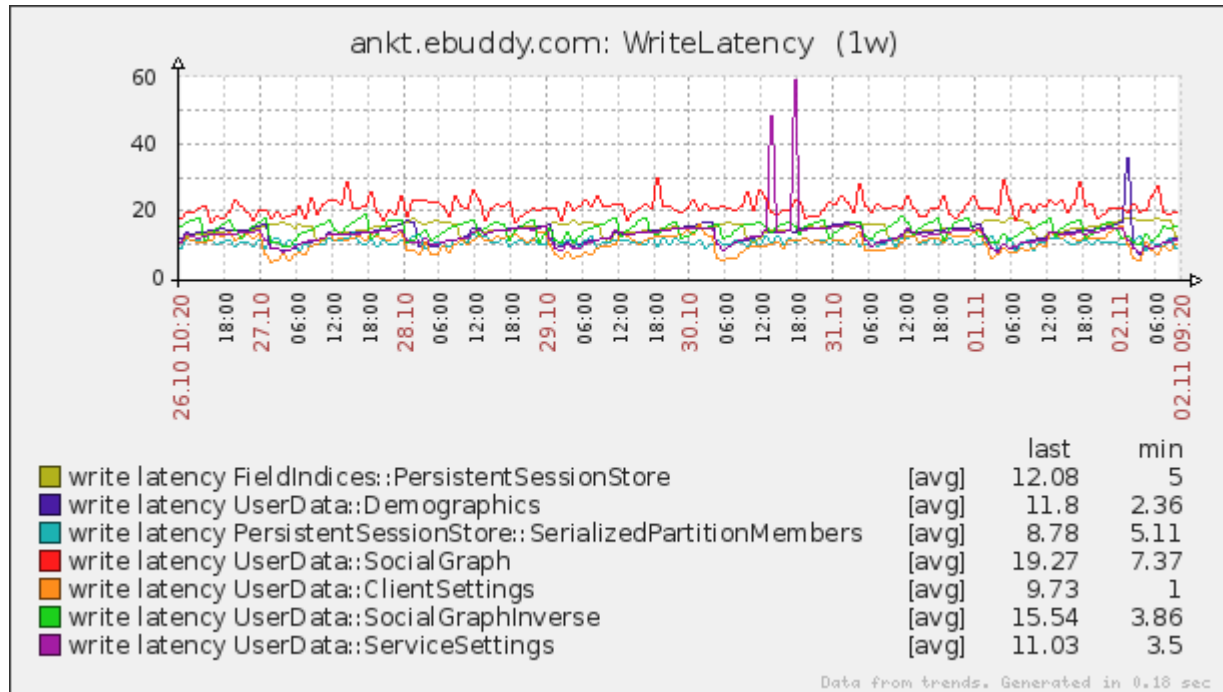


# Read Latency



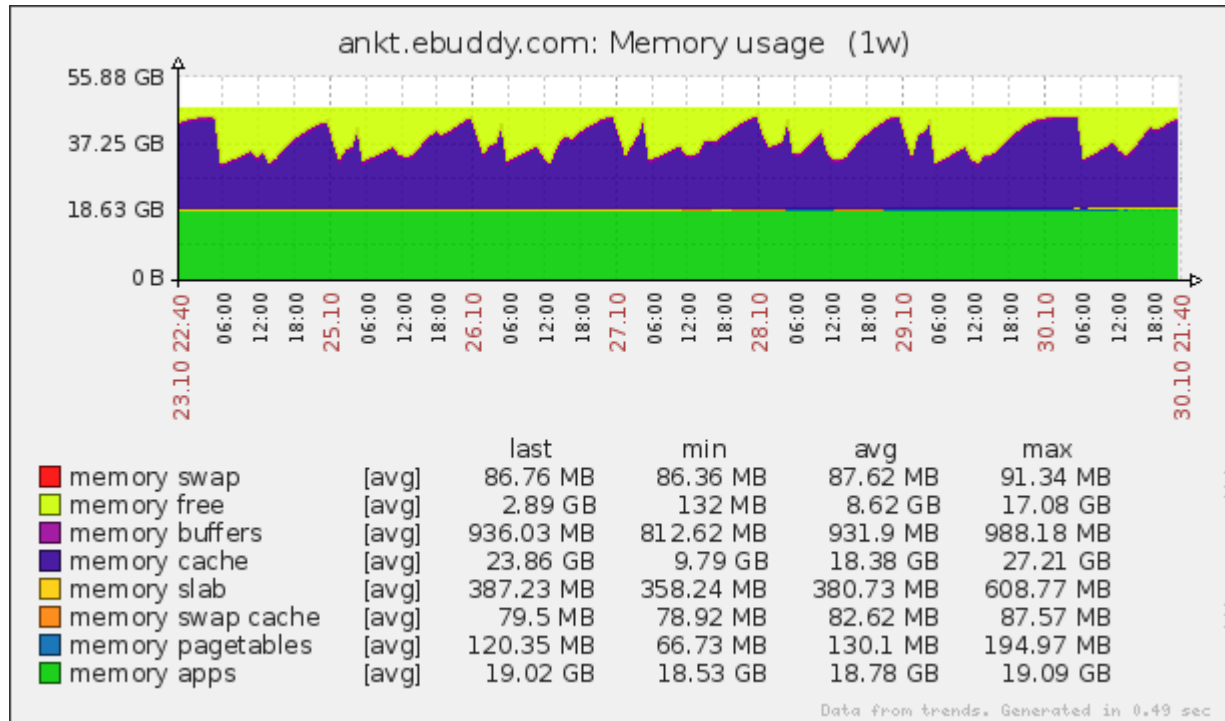


# Write Latency



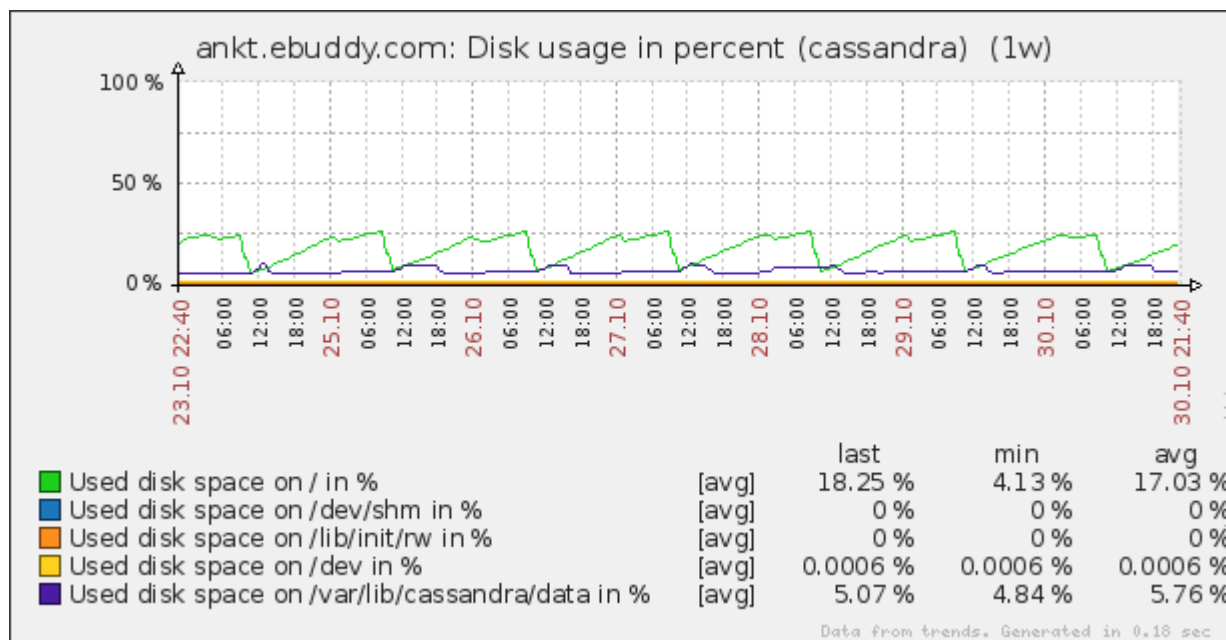


# Memory Usage



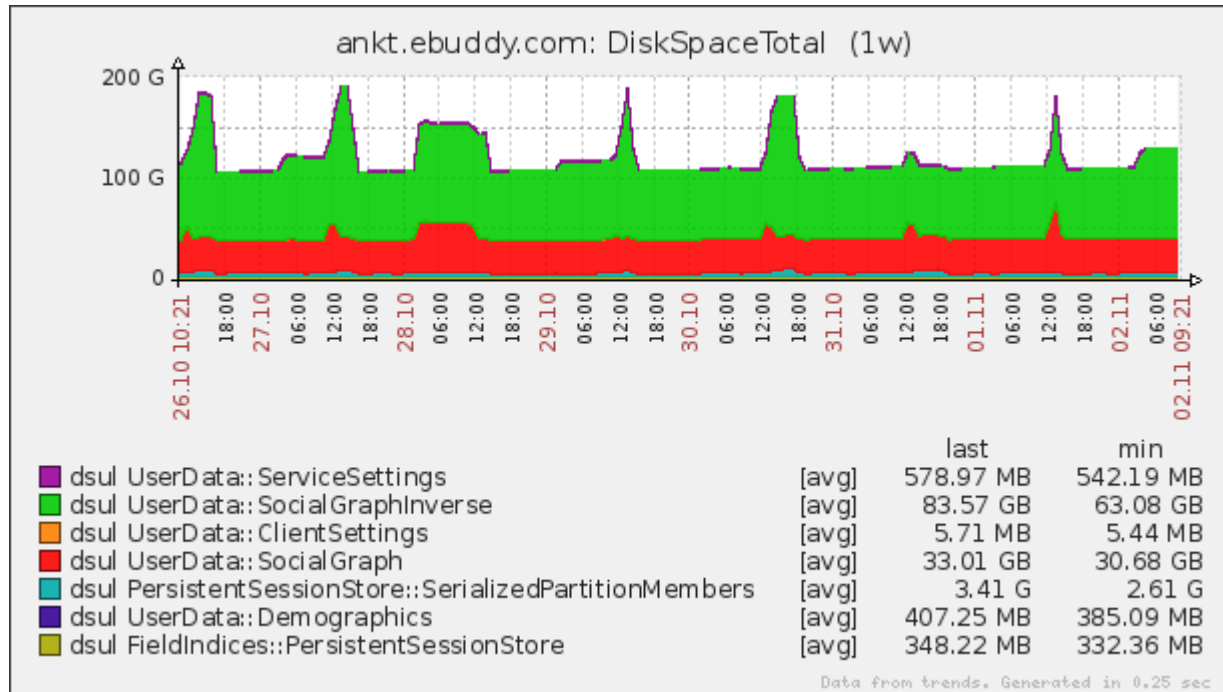


# Disk Usage



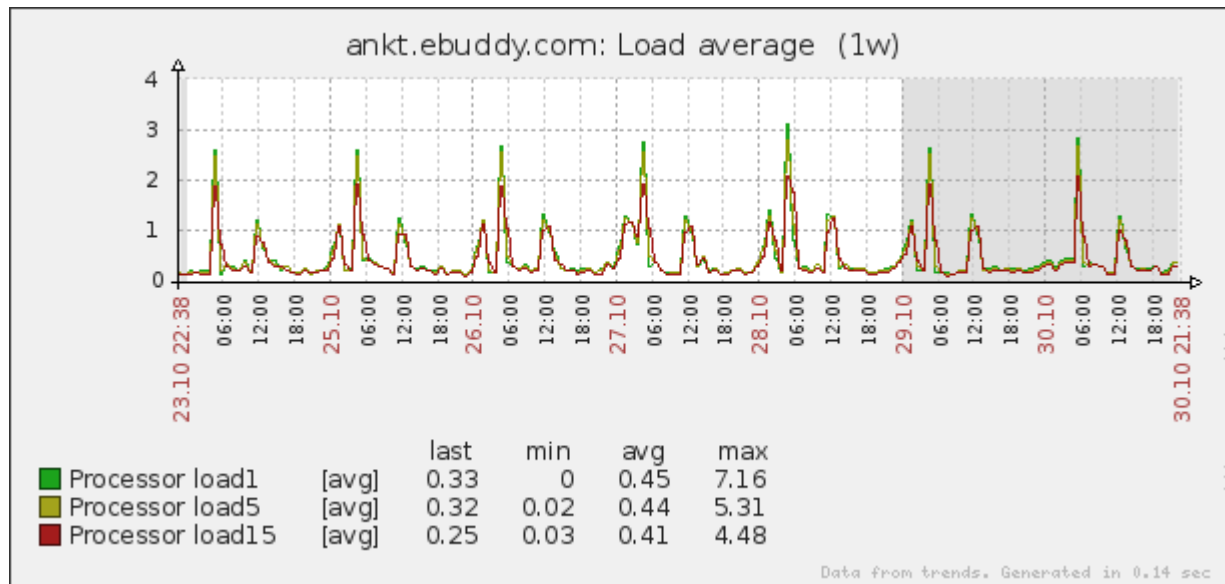


# Disk space total





# Load average





## What's next for Cassandra at eBuddy

- Upgrade to 0.8/1.0
  - Community Support 😊
  - Better read performance
- Start using (Dynamic)CompositeType
  - Create “deeper” structure without serializing JSON objects like we do now
- Add more use cases
  - Conversation History
  - IM application data



## What's next for Cassandra at eBuddy

- Look into Brisk
  - Integration with Hadoop/Hive
  - Real time Data Warehousing w/o ETL

.nl.  
jug



Q & A



- Try it out! (Android,iPhone,Blackberry)
  - <http://www.ebuddyxms.com/download>
- We're hiring (Mobile and Backend devs)
  - <http://www.ebuddy.com/jobs.php>
- Questions?